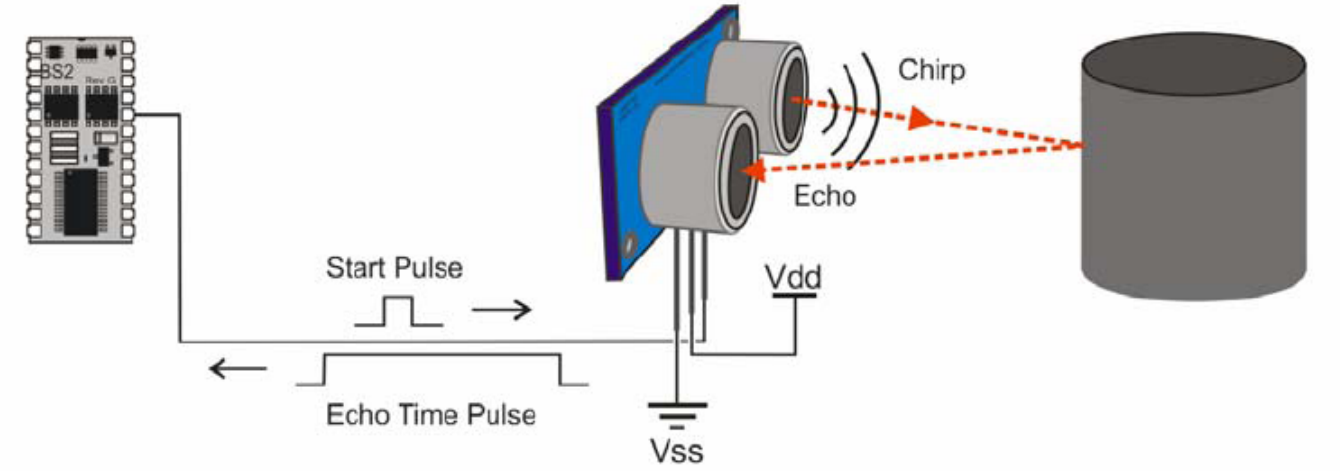
LAB 9

DISTANCE MEASUREMENT USING THE PING))) SENSOR AND TIMER INPUT CAPTURE

This week in lab, you will be working with the ultrasonic (PING) sensor and programming the General-Purpose Timer Module (GPTM) timer module in input capture mode to get distance measurements. You will use input capture to measure the width of an input pulse that is directly proportional to the distance measured by the PING sensor. Understanding the operation of the ultrasonic sensor and the GPTM timer module will be important for your goals in this lab, which will also be important for your goals with the project.

BACKGROUND

The range of the PING))) SONAR sensor is approximately 2 cm to 3 m, and it will only report one echo for any given chirp pulse. This one echo is the first received to meet the minimum threshold used to discriminate between noise and a proper echo. The chirp pulse does spread as it travels away, so the first reflection may come from an object that is not directly in front of the sensor. Clutter **does** affect the usefulness of the sensor.



SONAR (**SO**und **N**avigation **A**nd **R**anging) uses an ultrasonic burst (well above human hearing range) to determine the presence and distance of objects. SONAR is a term that is valid for both air and water. The frequency of the sound pulses emitted are different for these two applications. We will be implementing a primitive SONAR. The sensor emits 40 KHz pulses and estimates distance by using the fact that objects reflect sound. The distance is calculated by determining the time between emitting a sound pulse (chirp) and receiving an echo. The echo is translated into distance given the speed of sound in a particular medium. In our case, the medium is air. While air temperature does affect the speed of sound, for our purposes we will assume the speed of sound to be constant at 343 m/s.

The composition and shape of objects affects the amount of sound that gets reflected back to the sensor. A soft material like fabric will absorb more sound than a hard material like steel. It is

possible to angle a box so that sound waves will reflect away from the sensor, so the box may "disappear" like stealth aircraft.

The General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks. Each 16/32-bit GPTM block can provide two 16-bit (half-width) timers/counters: Timer A and Timer B. Thus, Timer A is a 16-bit timer, as is Timer B. These timers/counters can be further configured to operate independently as timers or event counters or concatenated together to operate as one 32-bit (full-width) timer. All timers have interrupt controls and separate interrupt vectors as well as separate interrupt handlers.

Each GPTM block can work in one of the following modes:

1. One-Shot Timer Mode
2. Periodic Timer Mode
3. Periodic Snapshot Timer Mode
4. Wait-for-Trigger Mode
5. Real-Time Clock Timer Mode
6. Input Edge-Count Mode
7. Input Edge-Time Mode
8. PWM Mode
9. DMA Mode
10. Synchronizing GP Timer Blocks
11. Concatenated Modes

In this lab we will be making use of the Input Edge-Time Mode (generally known as input capture), in order to capture the time a chirp pulse is sent from the PING))) sensor and the time that an echo is received back at the PING))) sensor. These times correspond to the rising and falling edges of the "echo time pulse" in the figure above. The PING))) sensor is connected to Timer 3B.

The ultrasonic chirp and its echo are sound waves generated and detected by the PING))) sensor. A start pulse from the microcontroller triggers the PING))) to generate the ultrasonic chirp. The PING))) sensor then waits for the echo bouncing back from an object. During the time that the sensor is waiting for the echo, it sends an echo time pulse to the microcontroller. The rising edge of the echo time pulse occurs when the chirp is emitted, and the falling edge of the echo time pulse occurs when the echo is detected by the PING))) sensor. Thus the width of the echo time pulse is proportional to the distance to the object.

REFERENCE FILES

The following reference files will be used in this lab:

- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- PING sensor datasheet - PING-v1.3-datasheet.pdf

- TI Tiva TM4C123G Microcontroller Datasheet
- Tiva TM4C123x Microcontrollers Silicon Revisions 6 and 7 Errata
- TI TM4C123G Register Definitions C header file: REF_tm4c123gh6pm.h
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- GPIO and GPTM register lists and tables: GPIO-GPTM-registers-tables.pdf
- Guide for using the PicoScope

The code files are available to download.

In addition to the files that have already been provided for you, you will need to write your own **ping.c** file and **ping.h** file and the associated functions for setting up and using input edge-time mode on the GPTM timer module. Template files have been provided. Separate functionalities should be in separate functions for good code quality and reusability. This means that in your ping.c file you should write separate functions for initializing/configuring the timer, activating the PING))) sensor, and calculating distance measurements. Remember to use good naming conventions for function names and variables. Minimally, we recommend defining the following functions:

```
void ping_init (void); //initialize the timer
void ping_trigger (void); //activate the PING sensor
float ping_getDistance (void); //calculate distance
```

You will also write an interrupt handler (interrupt service routine, ISR) for Timer 3B to read the timer values when the rising and falling edges occur on the PING echo time pulse. Remember, while you as a programmer write the ISR, you do not call it. It is executed automatically when a timer interrupt occurs and is processed by the interrupt system. Like with previous interrupt code, you will have volatile global variables to share information between the interrupt handler and other functions.

Your function prototypes may differ depending on the functionality you implement.

PRELAB

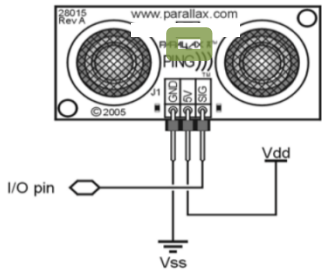
See the prelab assignment in Canvas and submit it prior to the start of lab.

STRUCTURED PAIRING

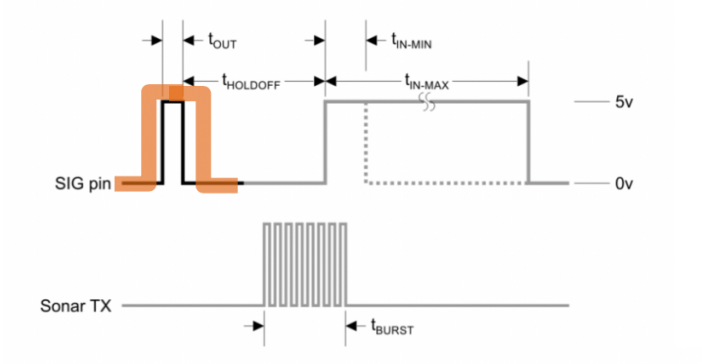
You are expected to continue to use structured pairing in this lab and in future labs. It was introduced in Lab 2.

PART 1: ACTIVATING THE SENSOR

The green LED located between the two cones on the sensor should blink at regular intervals when it has been activated, as shown in the figure to the right.



The sensor is activated, or triggered, with a short start pulse on its I/O signal (SIG) pin. The start pulse is a low-high-low signal, with a typical high time of 5 microseconds (t_{OUT} in the figure below, highlighted in orange). Can you find t_{OUT} in the PING))) datasheet?



Thus your first task is to trigger the PING))) sensor with a start pulse. This start pulse is sent by the microcontroller as a digital output on the GPIO pin connected to the sensor I/O pin, which is GPIO Port B pin 3, PB3. After the PING))) sensor receives the trigger, it will emit a sonar burst (chirp) (t_{BURST} in the figure).

To generate the start pulse, you need to initialize PB3 as a **digital output** (disable the alternate function) and then make the pin low-high-low, as pictured. This is done by writing 0-1-0 to the GPIODATA register bit. Refer to the PING))) datasheet for more information about the duration of the trigger pulse (how long it should be high) to meet the specifications of the PING))) sensor.

You will view the start pulse waveform using a probe connected to the pin. Refer to the PicoScope setup guide for more information. Launch the PicoScope program on the computer (find it from the Start menu).

CHECKPOINT:

Confirm that the PING))) sensor is activated by using the PicoScope program to display the start pulse. For this part, you are not yet required to read the sensor or calculate the distance. You will do that later.

PART 2: DETERMINE THE SENSOR ECHO TIME PULSE WIDTH

Now that the PING))) sensor is working, it will send an echo time pulse to the microcontroller (t_{IN} in the figure above), and your program needs to determine the width of the echo pulse (time duration between the rising and falling edges).

The input edge-time (or input capture) mode of the GPTM timer will be used to determine the pulse width. Refer to the preparation guide for the lab to understand this mode of the timer, reading-prep-TimerIC.pdf.

Specifically, for this lab, you will initialize Timer 3B as follows:

- input edge-time mode
- default 16-bit timer
- count-down direction
- use of 8-bit prescaler as an extension to get a 24-bit timer (need to load the PR and ILR registers with 0xFF and 0xFFFF, respectively)
- both rising and falling edges as input capture events being detected on the CCP pin
- interrupts are enabled for input capture events (an edge generates an interrupt)

We need to understand the wiring and functioning of the PING))) sensor. Notice there is only one I/O signal wire for the sensor. The same signal wire is used for both input to and output from the sensor.

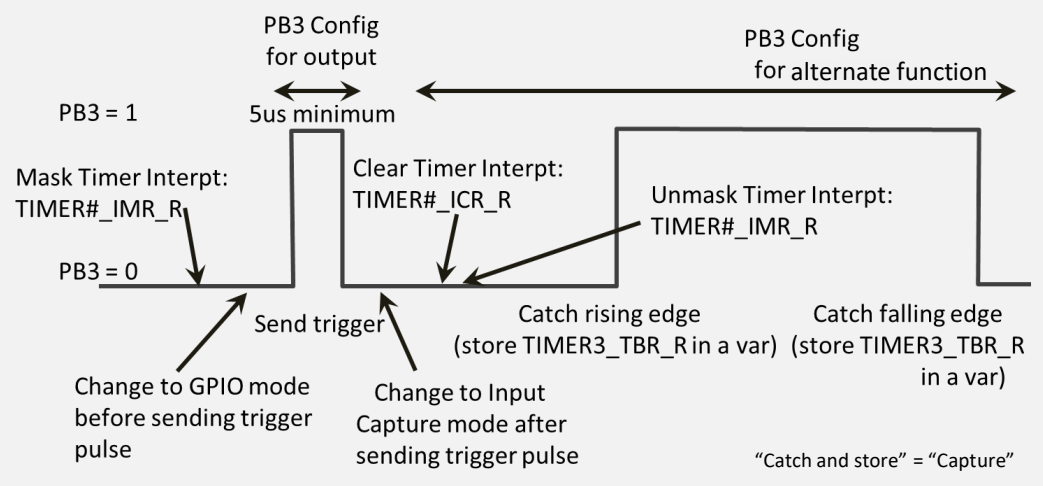
- In Part 1, you used the signal wire as the start pulse input to the sensor, and thus an output from the microcontroller to the PING))).
- In Part 2, you will be using the signal wire as the echo time pulse output from the sensor, and thus an input to the microcontroller from the PING))).

This means you need to reconfigure the use of the GPIO pin, PB3, in different parts of your code:

- In Part 1, PB3 was configured as a GPIO digital output.
- In Part 2, PB3 will be configured as a Timer 3B CCP input (alternate function T3CCP1). This configuration uses the GPIOAFSEL and GPIOPCTL registers. Note: There is no need to set the GPIO pin direction when it is used as an alternate function.

In other words, PB3 is initially configured as a digital output to activate the sensor (as in Part 1). Then it is dynamically reconfigured by your program as an input to the timer to detect the PING))) edges (this part of the lab). In this part of the lab, PB3 is used for both purposes to fully operate the sensor. This reconfiguration is done in pairs. Every PING))) sensor reading requires first using PB3 as a GPIO digital output to activate the sensor, then as timer input to get the pulse width.

You may want to sketch diagrams of the system, signals, data flow, and control flow to help you keep track of what needs to be coded. Start with the diagram below. Think about the configuration code and other code for each section of the diagram. What does the code need to do for each section? Where is the code written in your program (e.g., main, functions, interrupt handler)? What registers are used? Take it one section at a time. You already did the first section (trigger pulse) in Part 1.



IMPORTANT NOTE: You will need to modify the code from Part 1 to ensure that the timer, timer interrupt, and alternate function are explicitly disabled while you send the start/trigger pulse. The alternate function, timer interrupt and timer must be enabled before detecting the rising and falling edges of the CCP input and capturing the time values in the timer register. The timer must also be disabled while you initialize it. Carefully review the diagram above and the sample code provided. Pay special attention to the changes in timer configuration. If the timer is not properly disabled while the start pulse is being sent, we have observed erroneous interrupt behavior.

Note in the sensor datasheet timing specification, there is a guaranteed holdoff delay after the sensor receives the start pulse from the microcontroller and before it will respond with the rising edge of the echo time pulse. During this time, your microcontroller program must get ready to read the echo time pulse by reconfiguring the PB3 pin.

The echo pulse from the sensor will have a rising edge and then falling edge. The elapsed time between these two edges is directly proportional to the roundtrip distance between the sensor and the object. Input capture in edge-time mode will detect the edges of the signal and, when an edge is detected, store the current counter value into the Timer 3B timer register, GPTMTBR (TIMER3_TBR_R). The difference in counter values (rising edge to falling edge) represents the pulse width in clock cycles. The time in seconds can be calculated based on the system clock frequency, which determines the amount of time for each counter value.

Note the description of the GPTMTBR register in the Tiva datasheet (page 765):

> *In the Input Edge Time mode, this register contains the time at which the last edge event took place. ...*
> *In a 16-bit mode, bits 15:0 contain the value of the counter and bits 23:16 contain the value of the prescaler in Input Edge Count, Input Edge Time, and PWM modes, which is the upper 8 bits of the count. Bits 31:24 always read as 0.*

This means that, given the initialization of the timer to use the 8-bit prescaler as an extension to the default 16-bit timer, the value read from GPTMTBR is a 24-bit counter value.

For programming purposes, think about the state of the echo pulse (signal from the PING sensor) as initially low before the trigger; then after being triggered, there is a rising edge, thus high; and then after receiving the echo, there is a falling edge, thus done (until triggered again).

REMINDER: refer to the preparation guide for the lab, reading-prep-TimerIC.pdf. Go through the preparation guide, the prelab assignment, this lab manual, and the register list for the GPIO and GPTM modules (GPIO-GPTM-registers-tables.pdf), and double-check which registers you need to initialize and use. For example, refer to section 11.4.4 Input Edge Time Mode in the Tiva Datasheet.

Do not forget to include "driverlib/interrupt.h" so that you can call IntMasterEnable() to globally enable interrupts. Also do not forget to enable interrupts in the NVIC appropriately for Timer 3B.

Calculate the pulse width in clock cycles. Display this echo pulse time on the LCD in number of clock cycles. Vary the distance between an object and the sensor and observe the changing pulse widths. Select a pulse-width value written to the LCD and verify that it makes sense with respect to the echo pulse timing range for the PING))) sensor given in the datasheet.

What if you calculate a negative pulse width? Note that the timer is running continuously, so the counter values may span across the range boundary. In count-down mode, if the first value is smaller than the second value read from the timer register, this means the timer counted down to 0x000000, and then wrapped back to 0xFFFFFF. When this happens, a timer "overflow" occurs (the timer rolls over from all 0's to all F's). Assuming count-down mode and at most one overflow for any pulse width measurement, a negative pulse width indicates overflow. Is it okay to assume at most one overflow? The maximum 24-bit timer period is about 1 second in our configuration. Think about the typical and maximum echo pulse widths of the sensor. The overflow situation is more complicated if the pulse (or time between edge events) is longer than the maximum timer range or if there is more than one overflow. Convince yourself that your program can assume the simpler situation.

Remember, a negative pulse width doesn't make sense. Your program could discard it, or correct for it. Discarding it has some disadvantages. To correct for it, use the overflow condition to appropriately adjust the pulse width calculation. In this part of the lab, you can either ignore or correct for overflow. However, your program should display if overflow occurred.

CHECKPOINT:

Display the PING))) echo pulse width in clock cycles on the LCD and indicate if timer overflow occurred. Confirm that the pulse-width value makes sense. Try using a picoscope to display the sensor signal.

Errata for our microcontroller:

Interestingly, take a look at the Errata-TMC4C123.pdf document and the errata note, shown below. Due to error GPTM#11 in the errata, we are using the count-down direction when configuring the GPTM in input edge-time mode.

It is common for a complicated device, such as a microcontroller, to have known bugs (i.e., behaviors that do not match datasheet specifications). When companies discover these issues, they add them to what is called an errata document. This document will typically describe: i) the conditions that cause a given bug, ii) the behavior caused by the bug, and iii) how to work around the bug. The excerpt below is from page 34 of an errata document for our microcontroller. It specifies a configuration of the GPTM that should be avoided. The full errata can be found as one of the reference documents for this lab.

GPTM#11	***The Prescalar Does not Work Properly When Counting up in Input Edge-Time Mode When the GPTM Timer n Interval Load (GPTMTnILR) Register is Written With 0xFFFF***
Revision(s) Affected:	6 and 7.
Description:	If the GPTM is configured in Input Edge-Time count-up mode with the GPTM Timer n Interval Load (GPTMTnILR) register equal to 0xFFFF, the prescaler does not work properly.
Workaround(s):	Do not load 0xFFFF into the GPTMTnILR register when counting up in Input Edge-Time mode.

PART 3: CONTINUOUS DISTANCE MEASUREMENT

From part 2, you now have a program that calculates the pulse width using input capture. You will now use this pulse width to estimate distance to the object. First calculate the pulse width in milliseconds, and then calculate the distance in centimeters. Try to be accurate with your distance calculations, as you will be using them for navigation later on.

Although the range of the sensor is expected to be between 2 cm and 3 meters, experiments have shown the practical range of the mounted PING))) on the floor of the lab to be roughly 5 cm to 275 cm. Reflections from the floor are the likely cause for the maximum range being limited compared to the datasheet.

You should perform a distance estimation every 200 - 500 milliseconds, and continuously display the pulse width (in both clock cycle counts and milliseconds) and distance in centimeters.

Your program should correct for overflow and also display a running count of overflows that occur.

CHECKPOINT:

Display the PING))) echo pulse width in clock cycles and milliseconds, the distance to the object in centimeters, and a running count of the number of timer overflows.

PART 4: IR CALIBRATION USING THE PING))) SENSOR (OPTIONAL)

Because the PING))) sensor is fairly accurate in its distance measurement, it does not need to be rigorously calibrated similar to the IR sensor. As such, it can be used to provide a known distance to calibrate the IR sensor. For example, move the CyBot backwards away from a wall or object in the test field (e.g., use a foam board in the lab as a wall). Use your movement API, UART, and IR code.

Send data to the PC via PuTTY including the distance from the PING))) sensor and the quantized value of the IR sensor. Use this information to calibrate the IR sensor. Graph your results. The goal is that this will provide you with a quick calibration method for the lab project.

DEMONSTRATIONS:

1. **Functional demo of a lab milestone –** Specific milestone to demonstrate is Part 2.
2. **Debug demo using debugging tools to explain something about the internal workings of your system –** The TA will announce any specific debugging requirements at the start of lab; otherwise you will create your own debug demo based on your needs and interests in the lab.
3. **Q&A demo showing the ability to formulate and respond to questions –** This can be done in concert with the other demos.